(e.g., requests for instructions from portions of memory allocated for storing data).

Referring to FIG. **8**, in other embodiments of the present invention, the selector **50** is also implemented in hardware, and would likely be located in the same place as the hardware decoder **60**. The hardware instruction selector **50** stores the base addresses of the instruction streams that were loaded into memory and includes some logic to determine when to switch between instruction streams (e.g., a hardware timer or an instruction counter).

In one embodiment of the present invention an FPGA configured to implement the selector **50** is placed between the CPU **10** and memory **30**. The FPGA includes registers to store the addresses of the different instruction streams **32** stored in memory **30**. When a request to read an instruction is received from the CPU **10**, the selector **50** translates the read instruction into a request to read instructions from the selected stream by modifying the memory address to be read in accordance with the address of the stream stored in the registers. The selector would then forward that read on to memory **30** to read the currently selected stream.

In embodiments in which encoded instruction streams are used, the FPGA further includes registers for storing the encoding keys corresponding to the instruction streams and a decoder **60** configured to decode, using the stored encoding keys, the instruction received from the memory **30** before the instruction is sent to the CPU **10**.

In other embodiments, the registers, the selector **50**, and the decoder **60** are integrated into a CPU or a memory controller and configured to function in a substantially similar way, wherein the selector **50** is configured to intercept accesses to memory to be sent over the bus by redirecting those requests to one of the instruction streams.

Although various components of the selector **50** and the decoder **60** are described herein as being implemented in software or hardware, one of skill in the art would understand that various components and functionality could be implemented in software, hardware, firmware, and combinations thereof.

The following discussion provides a mathematical and experimental analysis of the effectiveness of synthetic processing diversity using multiple instruction sets as described above.

Generally, the likelihood of success of an attack decreases as the number of architectures (or instruction sets) increases. In a system in which an architecture is randomly selected from a plurality of architectures at application startup or when the machine is booted and under conditions in which the attacker knows the set of all possible architectures that the system could select from, the average number of attempts until a successful breach is equal to the number of architectures. As such, the likelihood of a successful attack is the inversely proportional to the number of architectures. Mathematically, the inverse relationship corresponds to a geometric distribution because each of the attack attempts can be viewed as a Bernoulli trial.

Given that x is the attempt the attack will be successful and p is the probability of success of an attack (fixed for a given set of architectures):

$$x=\text{geometric}(p) \tag{1}$$

As x is geometric, the probability that the k^{th} trial is successful is given as:

$$P(x=k)=(1-p)^{k-1}\times p \tag{2}$$

In addition, the expected value of a geometric distribution is given as:

$$E(x)=\frac{1}{p} \tag{3}$$

However, p is known (as discussed above, p=# of architectures^{-1}, so equation (3) reduces to:

$$E(x)=\frac{1}{\#\text{ of architectures}^{-1}}=\#\text{ of architectures} \tag{4}$$

To test this hypothesis, code injection attacks were performed against a virtual machine running within QEMU. In the experiments, the virtual machine was configured to select one of a plurality of unique architectures for each run. FIG. **9** plots these experimental results as the number of attempts until the first successful attack against the number of unique architectures in the virtual machine.

The graph shown in FIG. **9** agrees with what would be predicted by the results of the statistical analysis, as given in equations (2) and (5). However, these numbers assume that no action is taken by the system operator when these attacks fail. In embodiments of the present invention, when an attack fails, an invalid instruction alert can be displayed to the system operator (e.g., in an alert sent as a message popup on a display, as a notification on a mobile device, as an email alert, etc.) if the watchdog process detects an attack. Given the severity of instruction-level errors it is likely that a system operator (or security monitoring software/hardware) would recognize the incoming attacks and respond. In addition, the watchdog process may be configured to take preventative measures (e.g., lock down the system and/or prevent access to the system) after N unsuccessful attacks. As such, the probability that an attack is prevented can be expressed as:

$$P(\text{Attack prevented})=1-\sum_{0}^{N}P(x=i) \tag{6}$$

Equation (6) is using the geometric probably for each trial from (2). The watchdog approach was implemented in the experimental system for a number of 2 up to 47 architectures, with 25 runs experimental runs per data point and the results of the experiments are shown in FIG. **10**.

FIG. **10** illustrates the measured probability of preventing attack using a watchdog process for a given number of architectures as determined by from the watchdog runs. Once the number of architectures reaches 30, the odds for successfully stopping an attack begin to plateau at around 80% and additional architectures only marginally increase that percentage. The statistical analysis assumes a worst-case scenario; the attacker is aware of the set of architectures that could possibly be run, and that attack payload that will be used only needs to run in a single architecture.

However, in other embodiments of the present invention in which the architecture being executed switched during attack payload execution (e.g., when using multiple physical or synthetic architectures), then the attacker would also need to be aware of the change and construct the payload accordingly (e.g., such that the instruction set architecture of the injected